Simulating ROS at scale

Containers on Abel

Disclaimer!

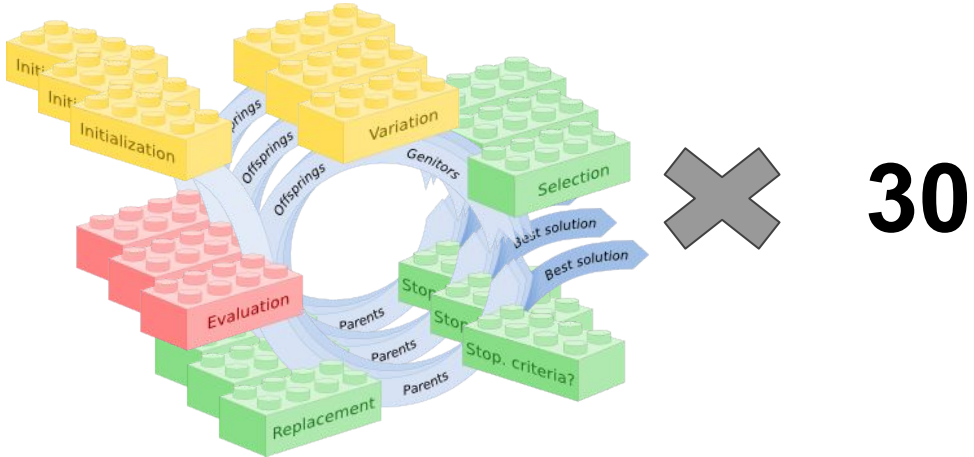
- Currently not working on Abel!
- Recipes depicted require `develop` branch of Singularity

- If interested contact me directly to get most up-to-date help =]

My motivation

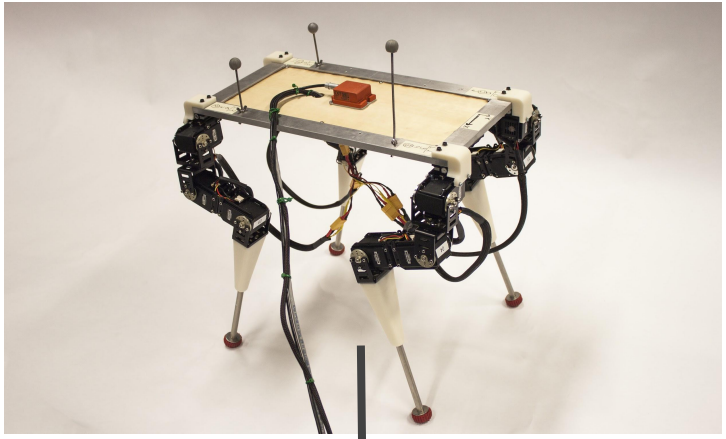
- Replication
  - In evolutionary robotics we need replication to gather *statistics*

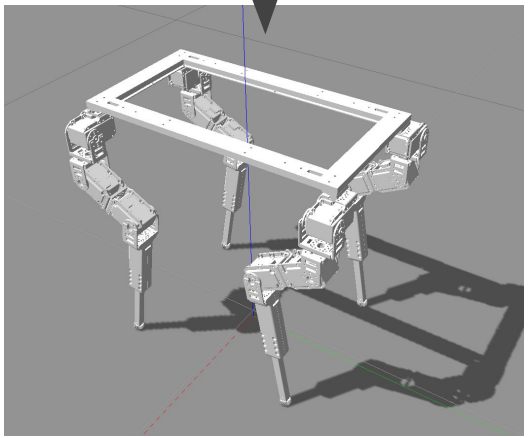
  - Current simulation runs *X* number of generations (8+ hours for 200)
  - Need to re-run simulation >= 30 times with 200+ generations



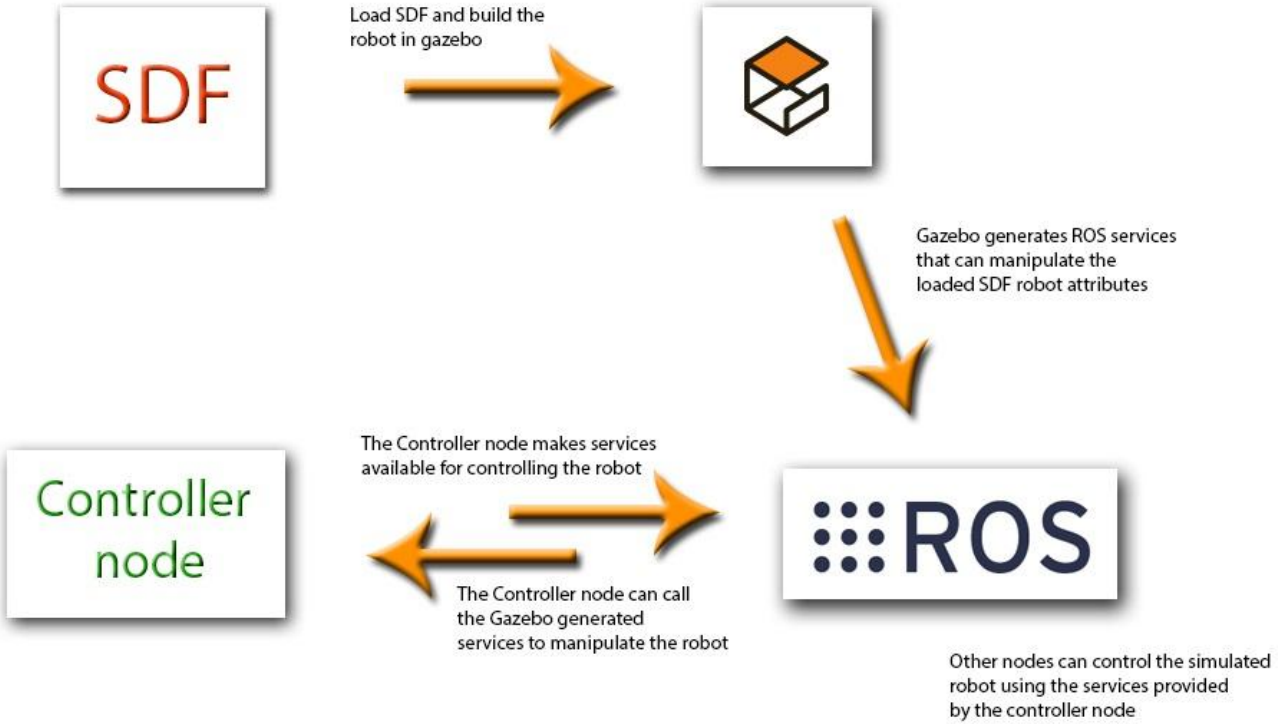
Simulating robots

With ROS

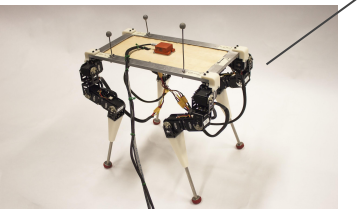




How?

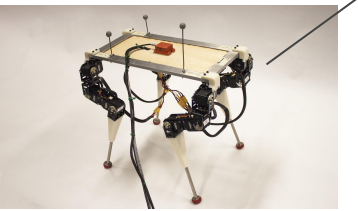

SDF
Load SDF and build the robot in gazebo
Gazebo generates ROS services that can manipulate the loaded SDF robot attributes
Controller node
The Controller node makes services available for controlling the robot
The Controller node can call the Gazebo generated services to manipulate the robot
ROS
Other nodes can control the simulated robot using the services provided by the controller node

How? - cont.



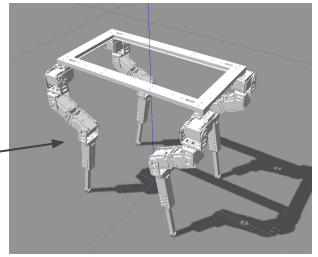
SDF
Load SDF and build the robot in gazebo
Gazebo generates ROS services that can manipulate the loaded SDF robot attributes
Controller node
The Controller node makes services available for controlling the robot
The Controller node can call the Gazebo generated services to manipulate the robot
ROS
Other nodes can control the simulated robot using the services provided by the controller node

How? - cont.



SDF

Load SDF and build the robot in gazebo



Gazebo generates ROS services that can manipulate the loaded SDF robot attributes

Controller node

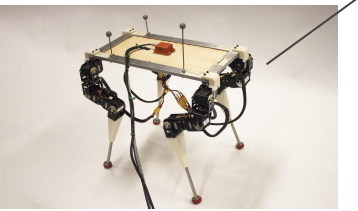
The Controller node makes services available for controlling the robot

The Controller node can call the Gazebo generated services to manipulate the robot

ROS

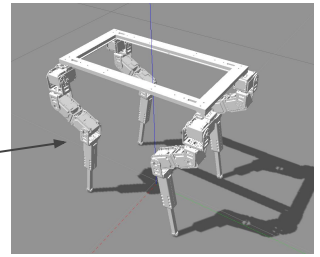
Other nodes can control the simulated robot using the services provided by the controller node

How? - cont.



SDF

Load SDF and build the robot in gazebo



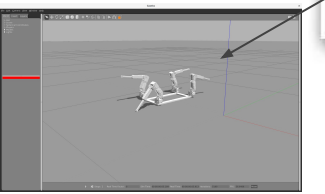
Gazebo generates ROS services that can manipulate the loaded SDF robot attributes

ROS

The Controller node makes services available for controlling the robot

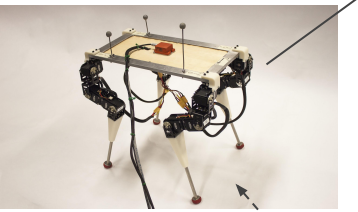
The Controller node can call the Gazebo generated services to manipulate the robot

Controller node



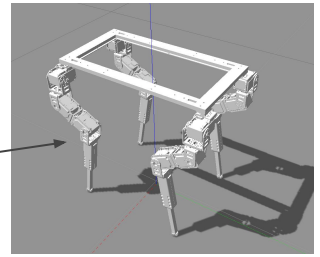
Other nodes can control the simulated robot using the services provided by the controller node

How? - cont.



SDF

Load SDF and build the robot in gazebo



Gazebo generates ROS services that can manipulate the loaded SDF robot attributes

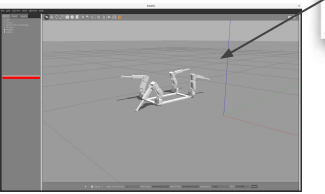
Controller node

The Controller node makes services available for controlling the robot

The Controller node can call the Gazebo generated services to manipulate the robot

ROS

Other nodes can control the simulated robot using the services provided by the controller node



Problem

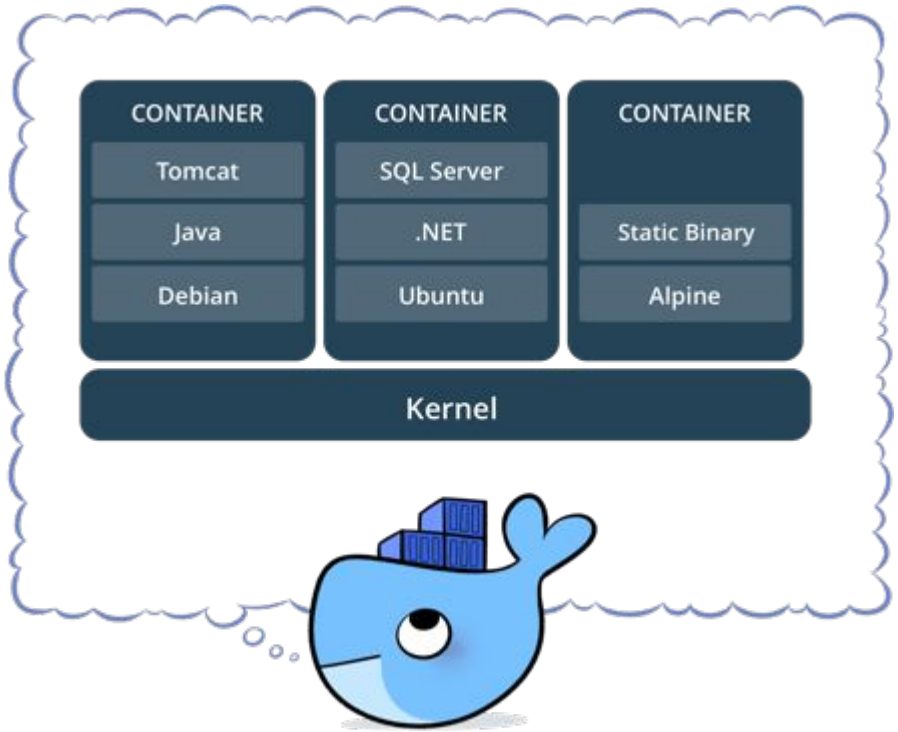
- ROS is “difficult” to run
  - Consisting of several, loosely, coupled packages
  - Loose coupling between components means difficulty in distributing
    - There is not a single executable that runs the simulation
    - Several individual processes needs to run
  - Runs *only* on Ubuntu
    - Abel runs RHEL

Why run ROS?

- ROS
  - Same controller for simulation and real-world
  - Access to all ROS packages
- Gazebo
  - Well tested
  - Several simulation engines “supported”
    - ODE, Bullet, Dart, SimBody

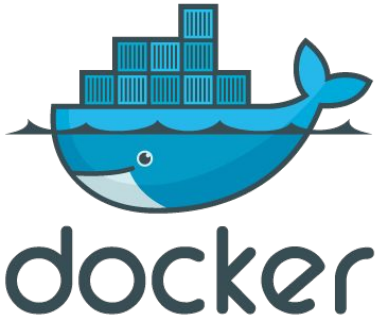
Solution?

- Package everything up into a container!
- Container can be passed around as *one* executable
- All necessary components packaged within container



Containers

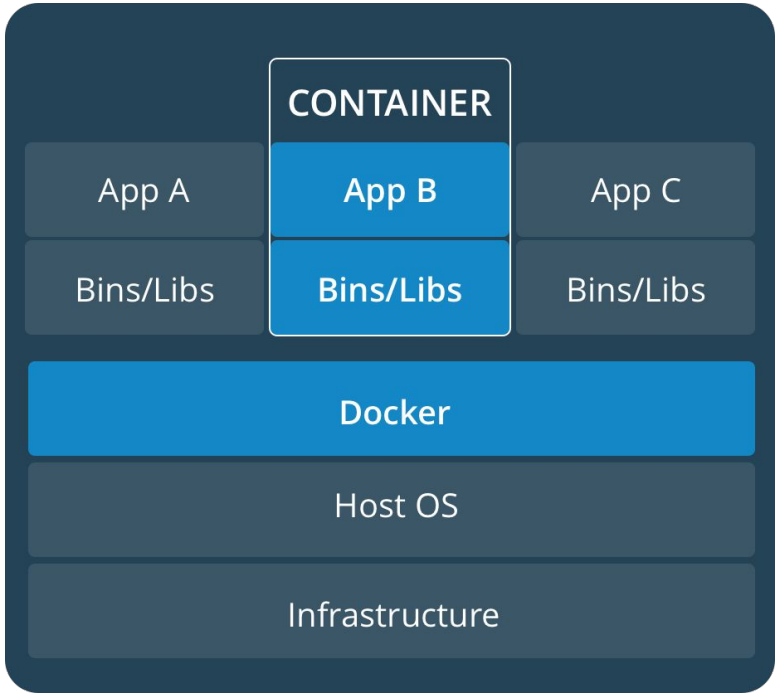
Packaging ROS simulations



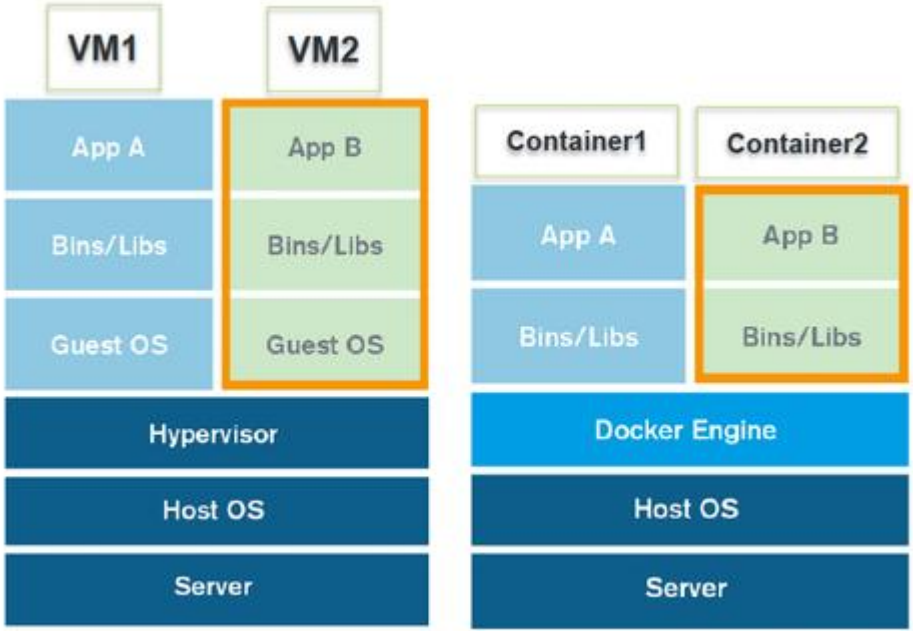
- What is a container?
- What is different?
- How to run it on Abel?

What is a container?

- Lightweight process similar to a Virtual Machine (VM)
- Contains everything your application needs



What is a container? - cont.



What is a container? - cont.

Advantages over VMs

- Simply a Linux/Windows process
  - Does not require full operating system
- As much as possible shared with host
  - Smaller in size compared to VM
  - Many services already started by host
- Fast startup
  - No need to boot a full operating system

What is a container? - cont.

Advantages in academia

- Reproducibility!
  - As long as the container system runs, the experiment will run
  - Experiment setup and configuration inside container
    - No version mismatch
      - Runs the same every time!
- Freedom from IT
  - Don’t have your required software, create container with the necessary components!

What is different?

- Almost nothing™
  - Develop software as normal
  - Once done or starting experiments -> package into container
    - Simple description of what is in the container
    - Start from several thousands of pre-built images
      - https://hub.docker.com/

Example

```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

Example - cont

Use pre-built Docker image

```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

Example - cont

Copy your code into container

```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

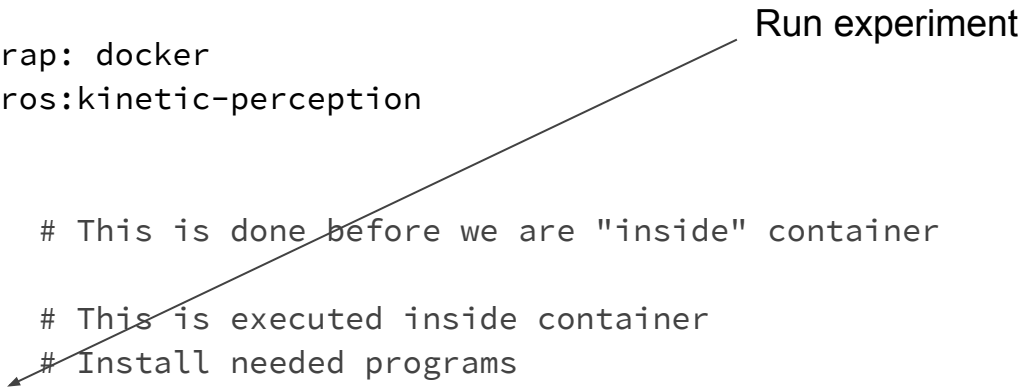
Example - cont

Install necessary libraries (from Ubuntu)

```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

Example - cont




```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

Example - cont

```
Bootstrap: docker
From: ros:kinetic-perception

%setup
        # This is done before we are "inside" container
%post
        # This is executed inside container
        # Install needed programs
%runscript
        roslaunch dyret_map_gaits exp1.launch
```

Once

Everytime

How to run it on Abel?

- Same as any other experiment
  - `singularity run image_name.img`
- Create repetition through “Array Jobs”

```
#!/bin/bash

# This is a job script for Abel meant to run several instances of the
# `job_script.sh` job script.

# Configuration for queue system
#SBATCH --account=uio
#SBATCH --mem-per-cpu=200M

# Setup environment on cluster
source /cluster/bin/jobsetup
# Clear inherited modules
module purge
# If anything fails we exit immediately
set -o errexit

# Run several replication of the `job_script.sh` script
arrayrun 1-3 job_script.sh
```

How to run it on Abel?

- Same as any other experiment
  - `singularity run image_name.img`
- Create repetition through "Array Jobs"

Create 3 instances of the container

```
#!/bin/bash

# This is a job script for Abel meant to run several instances of the
# `job_script.sh` job script.

# Configuration for queue system
#SBATCH --account=uio
#SBATCH --mem-per-cpu=200M

# Setup environment on cluster
source /cluster/bin/jobsetup
# Clear inherited modules
module purge
# If anything fails we exit immediately
set -o errexit

# Run several replication of the `job_script.sh` script
arrayrun 1-3 job_script.sh
```

How to run it on Abel?

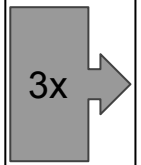
```
#!/bin/bash

# This is a job script for Abel meant to run several instances of the
# `job_script.sh` job script.

# Configuration for queue system
#SBATCH --account=uio
#SBATCH --mem-per-cpu=200M

# Setup environment on cluster
source /cluster/bin/jobsetup
# Clear inherited modules
module purge
# If anything fails we exit immediately
set -o errexit

# Run several replication of the `job_script.sh` script
arrayrun 1-3 job_script.sh
```




```
#!/bin/bash
# Configuration for queue system
#SBATCH --account=uio
#SBATCH --mem-per-cpu=1G
#SBATCH --cpus-per-task=2
# Shared variables:
IMAGE_NAME=dyret_map.img
RESULT_FOLDER=$SCRATCH/results_$TASK_ID/
# Setup environment on cluster
source /cluster/bin/jobsetup
# Clear inherited modules
module purge
# Loading singularity module
module load singularity
# If anything fails we exit immediately
set -o errexit
# Copy container image to work directory
cp $SUBMITDIR/$IMAGE_NAME $SCRATCH
# Setup result output directory
mkdir -p $RESULT_FOLDER
# Mark 'results' folder for copy once job is done
chkfile "$RESULT_FOLDER"
cd $SCRATCH
singularity run -B /work:/work dyret_map.img $RESULT_FOLDER
```

Further reading

- https://www.docker.com/what-container
- http://singularity.lbl.gov/ (Used on Abel)
- http://wiki.ros.org/docker/Tutorials/Docker (ROS Docker help)

- https://www.uio.no/english/services/it/research/hpc/abel/help/software/singularity.html (Abel help for Singularity)

- Contact me: Jørgen Nordmoen